Timely Result-Data Offloading for Improved HPC Center Scratch Provisioning and Serviceability

Henry M. Monti, *Student Member*, *IEEE*, Ali R. Butt, *Senior Member*, *IEEE*, and Sudharshan S. Vazhkudai

Abstract—Modern High-Performance Computing (HPC) centers are facing a data deluge from emerging scientific applications. Supporting large data entails a significant commitment of the high-throughput center storage system, *scratch space*. However, the scratch space is typically managed using simple "purge policies," without sophisticated end-user data services to balance resource consumption and user serviceability. End-user data services such as offloading are performed using point-to-point transfers that are unable to reconcile center's purge and users' delivery deadlines, unable to adapt to changing dynamics in the end-to-end data path and are not fault-tolerant. Such inefficiencies can be prohibitive to sustaining high performance. In this paper, we address the above issues by designing a framework for the timely, decentralized offload of application result data. Our framework uses an overlay of user-specified intermediate and landmark sites to orchestrate a decentralized fault-tolerant delivery. We have implemented our techniques within a production job scheduler (PBS) and data transfer tool (BitTorrent). Our evaluation using both a real implementation and supercomputer job log-driven simulations show that: the offloading times can be significantly reduced (90.4 percent for a 5 GB data transfer); the exposure window can be minimized while also meeting center-user service level agreements.

Index Terms—High-performance data management, HPC center serviceability, offloading, end-user data delivery, peer-to-peer.

1 Introduction

Modern high-performance computing (HPC) centers are charged with supporting scientific applications that increasingly use and produce very large data sets, e.g., analysis of neutron scattering data and deep-space observations. Of special importance are application result data sets or checkpoint snapshots from long-running simulations, which are required to be offloaded to end-user locations, where they can be analyzed for further scientific insights. For example, the Department of Energy's (DOE) Jaguar supercomputer at Oak Ridge National Laboratory (ORNL) (No. 1 in the Top 500 supercomputers) is generating terabytes of data from user jobs from a wide spectrum of science applications in Fusion, Astrophysics, Climate, and Combustion. Result outputs from Fusion applications such as GTC [1] and GTS [2] can reach up to 40 TB and 50 TB, respectively, for a $100{,}000+$ core run. In many cases, checkpoint data also doubles as result outputs that are used for inspecting job progress or for eventual aggregation into a set of visualized images (e.g., as in GTC, GTS, etc.). In fact, as the complexity and size of applications increase with the advent of petascale supercomputers, we may soon be faced with offloading a petabyte of data from a single application run. Another driving example is the TeraGrid collaboration [3], where result-data—from computations at any of the ten sites nation-wide—is required to be delivered to the end user. These user facilities are accessed by a geographically distributed user base with varied end-user connectivity, resource availability, and application requirements, delivering result-data to whom in a timely manner is a crucial challenge.

A common practice in HPC centers is to leave application-associated data management to the end user, as the user is intimately aware of the application's data needs. However, this approach ignores the interactions between different users' data demands, and their impact on center serviceability. To address this, in this paper, we focus on comprehensive end-user data management, which has largely been marginalized under current compute-focused center provisioning policies.

It is impractical to store all user data indefinitely at the center. The local storage in HPC centers, the *scratch space*, is used for job input, output, and intermediate data, which is typically on the order of terabytes. Scratch is built using a parallel file system that supports very high aggregate I/O throughput, e.g., Lustre [4] and GPFS [5]. To ensure efficient I/O and faster job turnaround use of scratch by applications is encouraged. Consequently, job input and output data is required to be moved in and out of the scratch space before and after the job runs, respectively. The scratch space requires proper provisioning to accommodate the storage demands of all incoming jobs, which in turn affects center serviceability.

HPC centers are aware of these constraints and enforce purge policies to manage the precious scratch space, wherein data is deleted based on a time window (ranging from a few hours to a few days) [6], [7]. As centers become crowded, the

- *H.M. Monti and A.R. Butt are with the Department of Computer Science, Virginia Polytechnic Institute and State University, 2202 Kraft Drive, Blacksburg, Virginia 24061. E-mail: {hmonti, butta}@cs.vt.edu.*
- *S.S. Vazhkudai is with the Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, PO Box 2008 MS6016, Oak Ridge, TN 37831. E-mail: vazhkudaiss@ornl.gov.*

Manuscript received 28 Oct. 2009; revised 2 Aug. 2010; accepted 25 Aug. 2010; published online 21 Oct. 2010.
Recommended for acceptance by F. Petrini.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-10-0534.
Digital Object Identifier no. 10.1109/TPDS.2010.190.

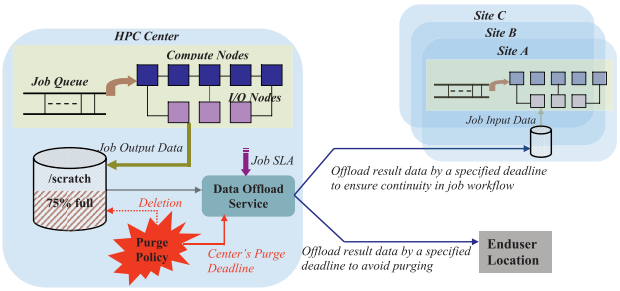

Fig. 1. Depiction of use cases for a timely offload of result data: (a) an expeditious offload to release center scratch space and to protect the data against a purge; (b) an end-user data delivery; and (c) data delivery to another part of the job workflow.

purge policies get more stringent to provide space for incoming jobs. The purge window is, therefore, a product of the center's load, its provisioned storage, and its desire to maintain a certain level of serviceability. However, there is no corresponding end-user service for a timely offload of data to avoid purging. As stated earlier, this is largely left to the user and is a manual process, wherein users stage out result-data using point-to-point transfer tools such as GridFTP [8], `sftp`, hsi [9], and `scp`. The inherent problem with using point-to-point transfer tools for offloading data from supercomputers is that they are only optimized for transfers between two well-endowed sites. For example, TeraGrid offers several optimizations (TCP buffer tuning, parallel flows, etc.) for GridFTP transfers between the various site pairs within TeraGrid, which are already well connected (10-40 Gbps links). The intent there, however, is to maximize the throughput between any two sites connected using state-of-the-art links. In contrast, end-user data delivery involves providing access to the data at the user's desktop. It cannot be ignored as a "last-mile" issue.

The lack of comprehensive result-data offloading affects not only end-user service, but also center operations. The output data of a supercomputing job are the result of a multihour—even several days'—run, and are usually stored in the center's scratch space. A delayed offload of such data results in sub-optimal use of scratch space in that the precious space is used for a job, that is, no longer running. Furthermore, a delayed offload renders output-data vulnerable to center purge policies. The loss of output-data leads to wasted user time allocation, which is very precious and obtained through a rigorous peer-review process. Thus, a timely offload can help optimize both center as well as user resources.

The need for timely data offloading is also driven by the, often, distributed nature of computing services and users' job workflow, which implies that data needs to be shipped to when and where it is needed. For example, several HPC applications analyze intermediate results of a running job, perhaps through visualization, to study the validity of initial parameters and to change them if necessary. A feedback loop is then employed, which involves tweaking the initial setup based on the newly acquired knowledge about the running simulation. This process entails expeditious delivery of the result data to the end user for online feedback. A slightly offline version of this scenario is a pipelined execution, where the output from one computation at supercomputer site A is the input to the next stage in the pipeline, at site B (see Fig. 1). Large-scale user facilities, such as SNS [10] and LEAD [11], which employ distributed workflows are already facing these problems and require efficient end-user data delivery services.

The common thread in both of the example cases above is the timely offload or delivery of output data. In the former use case, it can be stated as: *Offload by a specified deadline to avoid being purged*. In the latter, to: *Deliver by a specified deadline to ensure continuity in the job workflow*. In this paper, we design such a data service. The goal of this work is not to design a specific solution for capturing and streaming intermediate results. Instead, we provide an architecture for expeditious data delivery that can be used by existing tools.

1.1 Requirements of a Result-Data Offload Service

Current solutions for offloading large data to end-user sites are often mired by several factors. First, a direct download from the HPC center to the end-user requires that end resources be available for the entire duration of the transfer. This can be a significant space and bandwidth commitment from both the HPC center and the end user. For instance, the end-user resource might be unavailable when the data needs to be offloaded. This renders the result-data vulnerable to center purge policies. A desirable alternative, however, is to quickly move the data from center scratch space—perhaps to an intermediate storage location—so that the high-end, expensive resource can be relieved. Better yet, the intermediate location can be on the data path to the end user, so the data can be delivered from the intermediate location to the destination when the end resource becomes available again.

Second, current data offloading schemes from HPC centers do not exploit orthogonal (residual, unused) bandwidth that might be available between two transfer end points. Exploiting such bandwidth can help alleviate several problems endemic to data downloading, such as bandwidth volatility.

In essence, what is needed is an *architecture for timely end-user data delivery, that is, able to reconcile both the HPC center's as well as a user's constraints amidst varying bandwidth and resource availability conditions.*

1.2 Our Contributions

In this paper, we address the issues associated with providing a data-offloading service for HPC centers. Specifically, we make the following contributions.

1.2.1 Staged and Decentralized Offloading

We design a combination of both a staged as well as a decentralized offloading scheme for job output data. Compared to a direct transfer, our techniques have the added benefits of resilience in the face of end-resource failure and the exploitation of orthogonal bandwidth that might be available in the end-to-end data path.

1.2.2 User-Specified Intermediate Storage Sites

We adopt a novel variation to the use of intermediate sites (nodes) that differs from how they are used in most decentralized systems. The nodes participating in the transfer are specified and trusted by the user, thereby eliminating the concern of data delivery through a set of unreliable sites in a decentralized environment. We design ways, in which these nodes can be specified. Moreover, the design allows for using emerging resources, e.g., cloud nodes, as intermediate nodes as well.

1.2.3 Bandwidth Adaptation and on-the-Fly Decision Making

We develop a decision making component that factors in parameters such as a center's purge deadline, user delivery schedule and a snapshot of current network conditions between the center and the end user, to determine the most suitable approach to offload. We employ active monitoring, using the Network Weather Service (NWS) [12], to make the data offload process react to bandwidth degradation, thus ensuring that a user-specified delivery constraint or a purge deadline can be met.

1.2.4 Fault-Tolerant Offload

We utilize erasure coding schemes to ensure that the offload is fault tolerant.

1.2.5 Detailed Analysis and Evaluation

We have implemented the offloading service components and have thoroughly evaluated it using both trace-driven simulations, using a realistic simulator *simOffload*, as well as actual tests using the PlanetLab test bed [13]. *simOffload* is driven by three-year traces from the ORNL Jaguar supercomputer [14].

1.2.6 Integration with Real-World Tools

Finally, we have developed our solution in the context of real-world tools such as PBS [15] job submission system and BitTorrent [16].

2 Design

In the following, we first present an overview of the system architecture. Next, we discuss intermediate node selection and usage. Finally, we describe how individual components are integrated and utilized to provide the timely offloading service.

2.1 Architecture Overview

Fig. 1 illustrates the overall offloading framework, which entails a combination of strategies both at the center and the end-user site to orchestrate the transfers. The design challenges arise from the interplay between the center's purge policy, the job submission system and the data transfers for offloading.

We design a new software component, *Data Offload Manager*, to capture the above interactions and drive the offloading process. The *Manager* is integrated into the HPC center management software suite, and is provided with a number of critical center parameters and job descriptions to guide its operation. The *Manager* takes as input, guidelines regarding the purge deadline, D_{purge}, from the HPC center's scratch space purging system, and job specification from the job submission system. The specifications include the output data size, S, the job's data delivery schedule as per the Service Level Agreement (SLA), J_{SLA}, and other details such as any potentially available intermediate nodes, $<N_i, P_i, BW_i>$, where P_i denotes usage properties/constraints of the node, N_i, and BW_i denotes the current snapshot of the observed NWS bandwidth between the HPC center and N_i. Table 1 summarizes the list of parameters used in our system.

TABLE 1
Input Parameters for the Data Offload Manager

Parameter	Description	Source
D_{purge}	Purge deadline	Center configuration
S	Job output data size	Job specification
J_{SLA}	Data delivery schedule	Center-user SLA
$< N_i, P_i, BW_i >$	List, properties, and available bandwidth of intermediate nodes	Node discovery process

The *Manager* uses these parameters to determine a course of action, i.e., an offload schedule, O_s, for offloading the job's output data. O_s can be either a direct center to end-user site transfer or a decentralized transfer through the intermediate nodes. The goal is to deliver the data in time, $T_{offload}$, such that

$$T_{offload} \leq Min(D_{purge}, J_{SLA}). \quad (1)$$

Given the dynamic nature of the system, O_s needs to be constantly re-evaluated based on an updated $<N_i, P_i, BW_i'>$, where BW_i' is the latest snapshot of NWS measurements. Alternate routes have to be taken to meet the SLA if the re-evaluated time to offload, $T'_{offload}$, increases such that

$$T'_{offload} > J_{SLA}. \quad (2)$$

2.1.1 Parameter Specification

The offloading scheme relies on the job submission system for critical input parameters, some of which cannot be inferred from the center management software and must be specified by the end-user. For this purpose, we instrument the center's job submission system to enable end-users to provide information, e.g., delivery constraints and deadlines, etc., as part of their regular PBS [15] job scripts. The user simply submits the modified PBS script to our system on the center, which extracts the offloading-specific parameters and passes them to the offload manager.

2.1.2 Initiating the Offloading Process

Eager offloading has to be started to coincide job completion so that output data can be expeditiously staged out. Thus, a desired functionality of the job submission system is to be able to automatically initiate a prespecified process at job completion. We have done exactly that in our previous work [17], where, we instrumented the job submission system for starting user-specified direct data transfers, e.g., secure copy `scp` or GridFTP [8], upon job completion. This was accomplished by setting up separate queues for data and compute jobs, submitting the offload job to the data queue and specifying job dependencies such that the offload only begins after the compute job (dependency setup mechanisms are allowed by most modern resource managers). However, only simple user-specified direct data transfer commands were executed (e.g., scp or GridFTP) as part of offload in that work. In this paper, we use and extend our previous work to intimate the offload manager of the availability of a job's result data set for decentralized offloading of the data, which can then initiate the offload. The presence of a center-wide offload manager has the advantage that it can perform global optimization, for example assign a higher priority to an offload, that is, on a tighter deadline than others.

A final piece in the data offload architecture is the utilization of a number of user-specified intermediate storage locations or nodes to which data from the center is offloaded, and from which the end-user site can then asynchronously retrieve the data. These nodes are specified by the user as part of the job submission script. By selecting resources that are closer (in bandwidth) to the center, the offload bandwidth utilization can be maximized and the chances of loosing data due to a purge reduced. The intermediate nodes also provide multiple data flow paths from the center to the submission site, faster retrieval speeds, as well as fault tolerance in the face of failure.

2.2 Intermediate Nodes

In the following, we discuss the motivation, discovery, and utilization of intermediate storage locations in enabling a timely HPC data offloading process.

2.2.1 Motivation for Collaboration

The decentralized offload makes extensive use of intermediate nodes. We envision these to be nodes that are specified and trusted by the user. More specifically, consider the following collaboration scenarios that present a strong case for the participation of intermediate nodes in the data offloading process.

In today's HPC environment, supercomputing jobs are almost always collaborative in nature. In fact, a quick survey of jobs that are awarded compute time on the ORNL National Leadership Class Facility (NLCF)—through the DOE's INCITE [18] program—shows that these jobs involve multiple users from multiple institutions. This collaborative property is even more true in large national infrastructures such as the TeraGrid [3], that is, among the key drivers for end-user data delivery. Jobs in the TeraGrid are usually from a *virtual organization (VO)*, which is a set of geographically dispersed users from different sites, coming together to solve a problem of mutual interest for a certain duration. In such cases, it is clear that many users, from different sites will be interested in the resulting job output data. Thus, there is a natural need to dispatch the result data to more than a single location.

This property of collaborative science can be exploited to perform a collaborative offload of job output data. Participating sites can come together to form an overlay of intermediate nodes that contribute space and bandwidth for the offload. We argue that there exists a *natural incentive* for the participating sites to do so. Such a definition of intermediate nodes makes them more reliable and alleviates a key concern of precious result-data being transferred through an unreliable substrate.

The *natural incentive* works well when the project is a large collaborative one. In this case, our work does not expect any well-established infrastructure. Instead, it attempts to piggy-back on existing connectivity and residual bandwidth therein. Such a setup is well suited for long-running jobs where the overhead of intermediate-node setup is justified. More and more, we are observing that HPC jobs are either long running, or that the same users run a large number of small jobs, where the setup cost is amortized over the multiple runs.

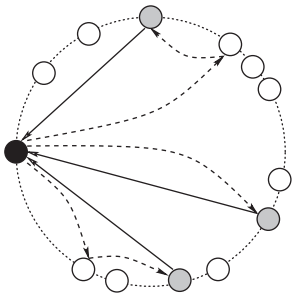

Fig. 2. Intermediate node discovery using random p2p messages. Here, the end-user submission site (black) discovers three intermediate nodes (gray).

2.2.2 Node Specification

The intermediate nodes are specified by the users as part of their job submission scripts. We provide special directives with which users can annotate their job scripts. These directives are parsed by the offload manager to extract and maintain a list of user-specified intermediate nodes. The explicit specification of all of the intermediate nodes that a user has access to may not be practical for large collaborations. Here, it is intended to demonstrate how a single user, even with just a handful of collaborating sites can exploit the intermediate nodes to conduct a collaborative download. In the case of large collaborations, we can imagine specifying simply the VO that the user is part of, in the job script. The data offload manager then submits the job to the scheduler. This way, the overlay of intermediate nodes becomes an integral part of the job and can be used for the delivery of the job's result data. End users can further qualify the intermediate node specification with usage policies, which specify the available storage and the load threshold at the intermediate node. For instance, an intermediate node might be willing to participate in the collaborative offload as long as the load incurred due to the transfer is below a certain level. We will discuss this specification in more detail later in Section 3.

2.2.3 Node Discovery

The intermediate nodes are selected from among the participating sites that are interested in the data transfer. However, not all nodes are available at all times. Thus, there is a need to discover appropriate volunteer intermediate nodes (N_is). Given the dynamic availability, and varied resource sharing policies of participants, a centralized approach would be cumbersome and impractical. Instead, we utilize the distributed and decentralized communication substrate provided by structured p2p networks [19], [20] to locate N_is in a dynamic environment.

We use a p2p overlay (Pastry [19]) to arrange sites that intend to participate in the collaborative offload (see Fig. 2). Use of the overlay provides reliable communication with other participants in the network. The participating sites, N_is use the overlay to advertise their availability to other nodes in the overlay using random broadcast. Nodes that receive these messages build local information about available nodes for offload. A given node can use its own policies and information about a remote node's capacity to make a decision regarding whether to use the remote node for the offload.

Finally, before submitting a job to the HPC center, the submission site, N_s, interacts with the center to sort the N_is with increasing latency from the center, while at the same time with decreasing latency from N_s. A greedy approach is sufficient here, as the dynamic nature of the system takes away any advantage of trying to further optimize such ordering before the actual offload process starts. The sorted set of nodes is provided to the center to utilize as the intermediate nodes, and becomes an integral part of the job's workflow.

While not part of our current implementation, the scalable p2p discovery can also exploit the resource discovery or selection mechanisms of a VO to identify intermediate nodes within a large collaboration. VOs typically use a Monitoring and Discovery System (MDS) that maintains a list of available resources that are willing to accept jobs. In our case, this infrastructure will need to be extended to accommodate storage resources willing to donate allocations and run our service.

2.2.4 Landmark Nodes

The reliance of our design on intermediate nodes exposes the offload system to possible failures due to lack of sufficient N_is. For instance, the submission site may not have access to any (or sufficient enough) intermediate nodes on the path to the HPC center. This could be either due to the lack of many participating sites in the job or due to the volatility of the intermediate nodes. To avoid such a scenario, we utilize a number of geographically distributed Landmark nodes that are always available and can serve as intermediate nodes in case enough p2p-nodes are not available. The Landmark nodes can be other HPC centers, or nodes along national links such as, Internet2 [21] Lambda Rail [22], REDDNET [23], or the TeraGrid [3] to which many end users may be connected and have access to. The location and number of the Landmarks is determined through out-of-band agreements with the HPC center. An example application for this use case is CERN's LHC [24] experiment, which is proposing to use national and regional sites as Tier 1 and Tier 2 data distribution centers to disseminate the experimental data from Tier 0 at CERN. Individual users can download data from these tier sites depending on geographic proximity.

2.3 The Data Offloading Process

The offloading process is initiated at the completion of a job as follows: First, the center chooses a number of nodes from the set of N_is ordered by available bandwidth. The exact number of nodes used for this purpose, i.e., the fan-out, is chosen to achieve maximum (prespecified) out-bound center bandwidth utilization, or to meet previously agreed-upon offload deadlines. These chosen N_is serve as the Level-1 intermediate nodes. Note that the selected fan out is not static, and can vary depending on the transfer speeds achieved. Second, the result data is split into chunks and parallel transfer of the chunks to Level-1 nodes is initiated. Since the Level-1 nodes are much closer to the center than the submission site, the offload time is expected to be much smaller than a direct transfer to the submission site. If not, the manager would have opted for a direct transfer schedule to the end-user site, and not the decentralized offload. This has the desired effects of both releasing the precious scratch space occupied at the center

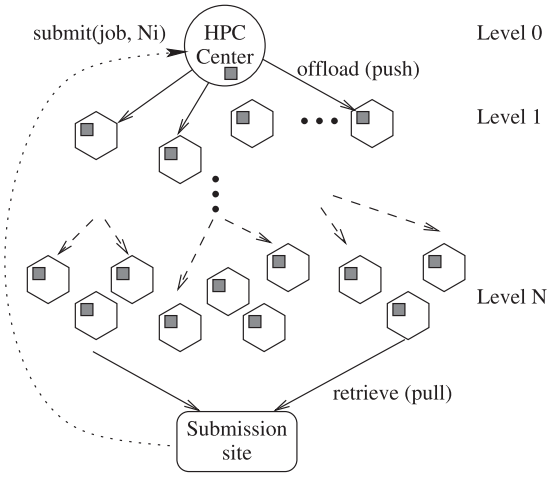

Fig. 3. The data flow path from the HPC center to the submission site. The intermediate nodes are represented by hexagons. The participants also run an instance of the NWS (gray square) for bandwidth monitoring.

and protecting the data from the purge. Third, Level-1 intermediate nodes may also further transfer data to the Level-2 intermediate nodes (once again chosen from N_is), and so on. Consequently, data flows toward N_s, though it is not pushed to N_s. Finally, N_s can asynchronously retrieve the data from the Level-N nodes. Decoupling N_s from the data push path allows the center to offload the data at peak (prespecified) out-bound bandwidth without worrying about the availability (and connection speed) of N_s, while enabling N_s to pull (retrieve) data from N_is as necessary. The key steps in the offload process are illustrated in Fig. 3.

The Push of data from one level to another (e.g., Level-1 to Level-2) is similar to the initial offload process, and is decentralized. Similar to the center, Level-i nodes may want to achieve a predetermined out-bound bandwidth, or may simply be configured to offload the data they have to a configurable number of Level-$(i+1)$ nodes for replication purposes. Either option results in choosing the fastest nodes to complete the Push operation.

The use of intermediate nodes in our system provides multiple data-flow paths from the center to the submission site N_s, leading to several alternative options for data delivery. For instance, data may be replicated across different N_is during the transfer from one level to another. This will allow N_s to pull data from a number of locations, thus providing fault tolerance against node failure, as well as better utilization of the available in bandwidth at N_s. The schedule can also be used to simultaneously deliver data to multiple interested sites.

2.3.1 Providing Service Guarantees

The submission site and the HPC center have SLAs regarding how quickly data can be offloaded from the center. Similar to the intermediate node specification, the SLAs are also specified in a job script.

Given, the dynamically changing bandwidths between participants, a fixed, or statically chosen fan out is insufficient. Therefore, we utilize a bandwidth monitoring-based scheme to dynamically adjust the fan out and ensure meeting the SLA. For this purpose, we employ the NWS [12] to monitor and estimate the available bandwidth

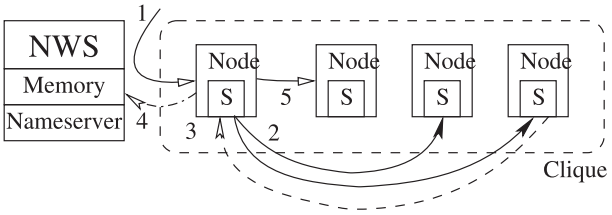

Fig. 4. Bandwidth monitoring using the NWS. "S" indicates our software.

between participating nodes. As seen in Fig. 4, each participating node joins a "clique," which is a group of sensors that measure bandwidth. A token is passed around (Step 1), which serves as an indication to a node to probe (Step 2) other nodes for available bandwidth. The replies (Step 3) are recorded not only at the node, but also at a central NWS repository (Step 4). The token is then forwarded to the next node (Step 5). The clique gives the center an estimate of the bandwidth available from it to different nodes. The center uses this information to decide whether a chosen fan out is sufficient to meet a particular SLA, or needs to be increased. If needed, additional nodes from the set of N_is can be chosen to increase the fan out and meet the SLA. Nodes at Level-i utilize a similar approach to determine the fan out for Level-$i+1$. At each level, a decision making component re-evaluates the time to offload as mentioned earlier. In case the number of available N_is are insufficient for meeting the SLA, the submission site is informed, which in turn can either provide more intermediate nodes or accept the best effort from the HPC center.

2.3.2 Fault Tolerance through Erasure Coding

As stated earlier, pieces of the result data can be replicated across many participating intermediate nodes, facilitating retrieval from any subset of the nodes. In addition to this, we apply erasure code [25], [26] to the data to improve the reliability of the transfer, while minimizing the amount of transferred data. The computational cost of erasure coding can be paid by the Level-1 intermediate nodes if coding at the HPC center (which will be part of the job's time allocation) is an issue.

1) Discussion: Recent studies have shown the high rate of storage system failures [27], [28], [29], and the complexity of ensuring reliability in large-scale installations [30], [31], [32] such as the HPC scratch space. Improving reliability in such fixed installations entail going through a rigorous and time-consuming acquisition process mired with delays. In contrast, the collective use of less-reliable individual intermediate nodes can provide a solution that can be arbitrarily grown to accommodate any desired level of reliability. Thus, we argue that although individual intermediate nodes may be more prone to errors compared to single disk in an HPC center, as a system our approach is able to provide better reliability due to its flexibility. Plus, this reliability comes for free as we use resources volunteered by collaborators, which would; otherwise, not be used [33].

2.4 Design Summary

By way of eagerly offloading result data from the center, our system avoids data loss due to center's purge policies. This in turn allows the center to free up precious scratch space for incoming jobs and their data, thereby improving its serviceability. By staging the data on an intermediate network of nodes, enroute to the destination, we ensure that the offload will not fail due to end-user resource unavailability. The result data can be pulled from the intermediate nodes as and when the end-user resource becomes available. Finally, our design provides an integrated data management solution for the HPC center, rather than leaving it up to the users, thus allowing them to focus on their applications and not bogged down by unnecessary system-level details.

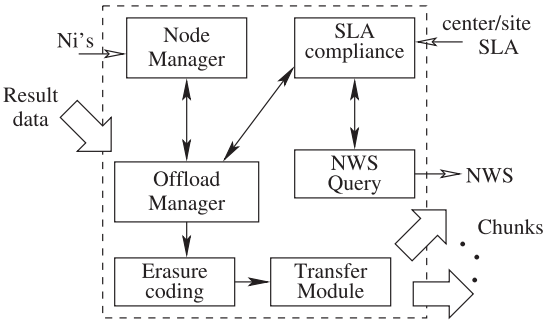

Fig. 5. The per-node system components and their interactions.

3 Implementation

The implementation of the offloading framework comprises of about 3,000 lines of C code, with the p2p substrate built using FreePastry [34] in Java. Fig. 5 shows the architecture of the software that runs on all the participating nodes. The software also runs on the HPC center as an *Offloading Service*. The list of N_is and the SLA are provided through the job submission script. The role of various components is as follows: The *Node Manager* is responsible for maintaining N_is. The *SLA Compliance* module uses bandwidth predictions provided by NWS [12] (through the *NWS Query* module) to guide the offload process in meeting that SLAs. The *Erasure coding* module transforms the data to be sent out into error-coded chunks, and the *Transfer Module* is charged with pushing out the encoded chunks to the next-level intermediate nodes. Finally, at the heart of the system is the *Offload Manager* that integrates all the modules and uses them to select different offload schedules and to enable the transfers. The erasure code that we have used is Reed Solomon (RS) [35] in 4:5 coding configuration, i.e., four input chunks are coded to produce five output chunks, with a redundancy of 25 percent. The chunk-size is a tunable parameter, which can be set based on the size of the data sets being offloaded.

3.1 Integration with Job Submission System

HPC centers utilize job management systems, e.g., batch job queuing using PBS [15], to ensure proper operation. Typically, the job submission system constitutes a user job script and a resource manager at the supercomputer center that schedules the jobs based on a queuing system. Thus, the natural place to specify user-defined intermediate nodes and deadlines is the existing job submission scripts.

To this end, we have instrumented the PBS [15] job submission system that is prevalent in HPC centers to

TABLE 2
New Script Directives Used for Offloading

Directive	Parameters	Description
`Stageout`	Output dataset, destination site	Specifies offloading dataset and the target destination site
`InterNode`	IP address, bandwidth snapshot, availability, capacity	Specifies an intermediate site location
`Deadline`	Time	Specifies the deadline for the offload to complete

enable specification of user-defined intermediate nodes and deadlines. We have devised a way for specifying intermediate nodes and delivery deadlines as annotations within a standard PBS script. These annotations are specified as directives, much like other PBS directives (e.g., `#PBS`). The intermediate nodes can be further qualified with policy specification that captures usage constraints. These constraints include the amount of space available for offload on a node, and the node's availability. More fine grained policies can be easily added.

Table 2 presents the directives that we have introduced to support the offloading process. Fig. 6 shows an instrumented PBS script with these directives, wherein a user specifies the stage out to a destination, the use of intermediate nodes with their space constraints, a port number where our transfer protocol is listening, and a delivery deadline.

To handle the instrumented job script, we have implemented a parser that runs on the HPC center. When an annotated PBS script is submitted for execution to the job scheduler at the HPC center, it is intercepted by our parser that filters out directives specific to data offloading, and passes those details to the *Offloading Service* for data delivery. The remaining PBS script is then handed over to the PBS queue for standard processing. As discussed above, the *Offloading Service* is aware of the center's purge deadline and attempts to reconcile that with user delivery deadline and intermediate/landmark nodes to achieve a desired data transfer schedule.

3.2 Integration with BitTorrent and NWS

We have designed our offloading mechanism to exploit the data dissemination abilities of BitTorrent [16] and network monitoring facilities of NWS [12]. While both of these services are centralized in our current implementation, we note that the design provides for distributed equivalents to be built and substituted easily.

Each participating node in our system runs an NWS daemon. We have configured NWS sensors that keep track of the vital statistics of each node, as well as record bandwidth measurements between nodes. These measurements are retrieved by our *Offload Manager* via periodic queries and used in determining appropriate offload paths

```
#PBS -N myjob
#PBS -l nodes=128, wtime=12:00
mpirun -np 128 ~/MyComputation
#Stageout Output DestSite
#InterNode node1.S1:49665:50GB
...
#InterNode nodeN.SN:49665:30GB
#Deadline 12/14/2010:12:00
```

Fig. 6. An instrumented PBS script.

that can sustain sufficient bandwidth to meet specified SLAs. The *Offload Manager* also employs the data from NWS to select additional peer nodes in case an SLA cannot be met.

The decision to add additional nodes to the offload path is driven by several factors: user-center delivery and purge deadlines, storage capacity of nodes (specified via the PBS script), and the available bandwidth.

Once a set of intermediate nodes is selected using NWS, we use BitTorrent's scatter-gather protocol to transfer the file from the center to the selected intermediate nodes. The offload happens as follows: The *Offload Manager* creates a metadata "torrent" file for the subset of data to be transmitted to a set of chosen intermediate nodes. The Manager also provides BitTorrent *tracking* services so that the intermediate nodes may know what data has been transmitted to which node. Once the nodes receive the torrent file, they use the metadata information along with the tracker to "download" the data subset to their local storage. The process is repeated at all the intermediate node levels. The end host can also use appropriate torrent files to download the result data from the intermediate nodes, thus completing the offloading process. Finally, issues that could arise due to the use of multiple data sources are simplified by using BitTorrent. For example, if two Level-1 nodes decide to send the same data set to a Level-2 node, BitTorrent will automatically utilize both copies of the data at the Level-1 nodes to quickly complete the transfer.

3.3 Deployment

We briefly highlight some deployment issues pertaining to the design details illustrated above. Earlier, we discussed how, given a set of intermediate nodes, our approach can discover a subset of them and compose them in a scalable fashion for a collaborative data delivery. However, collaborative the nature of scientific discovery, in day-to-day supercomputing environments, resource sharing often boils down to agreements between compute clusters, storage resources, and networks. The Grid community has spent a significant amount of time and effort in enabling these policies and collaborations, and we can leverage much from it. Instead, in this brief deployment discussion, we put forth the concept of a *Storage Service*, a piece of software that an intermediate node can run to participate in our infrastructure. The storage service is essentially the building block for constructing our overlay and involves an intermediate node allocating a certain amount of storage (exposed at a mount point), advertising the protocols available for data transfers, and installing and running the software necessary for scalable discovery. The service also runs our BitTorrent-based data servers and clients as well as an NWS daemon. A node can choose not to run our data movement tool and instead opt for an existing transfer tool. Our data delivery mechanism will need to factor this, in addition to the

advertised available storage, into the decision making. In the future, we can envision a catalog of such storage servers be maintained at a well-known location in the case where the user is part of a collaborative science team. In those cases, users need not specify the intermediate nodes in their job script as that can quickly become cumbersome.

4 Simulating the Offloading Process

The interplay between the different system components at an end-user site and the HPC center is complex and requires a controlled environment for in-depth analysis, which is near-impossible to do in real HPC setups. Thus, we have developed a realistic simulator for the offloading process, *simOffload*, which models both job execution and data offloading.

1) Job Scheduling: In *simOffload*, jobs are scheduled using a First-Come First-Served (FCFS) policy with back filling. Here, a number of large jobs are first scheduled in the order they arrive, until a majority of the machine's resources is allocated. Next, smaller jobs are scheduled. This approach is often employed in large-scale super-computers, such as Jaguar [14], which are intended to run a few large jobs that take up most of the machine (e.g., 100,000 cores). However, such larger jobs can leave a small but significant number of cores idle; back filling helps to avoid this by assigning smaller jobs to the idle cores. The goal is to strike a balance between the HPC center's desire to cater to "hero apps" that could take up an entire machine and potential idle cores.

2) Trace-Driven Simulation: simOffload utilizes a number of different traces to provide an accurate model of the system. Specifically it uses job and bandwidth traces.

a. *Job Traces:* The job traces were obtained from ORNL's Jaguar supercomputer [14] and represent nearly three years of job execution [36]. These traces provide for each job: arrival time, start time, total job execution time, and the compute resources used. Additionally, the traces also contain the amount of physical memory and virtual memory used by a job. The memory values and compute resources are used to estimate the amount of data produced by a job, e.g., the product of the available memory per core and the number of cores requested by an application provides the size of a possible checkpoint, which needs to be offloaded.[1] Finally, each job in the trace corresponds to a job executing and offloading in our simulator.

b. *Bandwidth Traces:* We model the intermediate nodes by using NWS bandwidth measurements from 50 different sites on the PlanetLab [13] test bed. The bandwidth traces provide pairwise bandwidth measurements for the 50 sites over a duration of 96 hours. Each simulated node in *simOffload* is assigned a measured trace. Since there are more nodes in the simulator than measured on PlanetLab, some nodes will have duplicate bandwidth traces. Nodes running for longer than 96 hours simply loop through their associated trace. Since the measured bandwidth is for pairwise exclusive communication, it does not capture the behavior when a node is participating in multiple transfers. In *simOffload*, we make an assumption that the available pairwise bandwidth is reduced proportionally to the number of offloads, in which a node is involved.

1. HPC centers neither log the submission scripts, nor the input and output data generated by specific job. It is not possible to change this behavior due to administrative reasons. Thus, we have to resort to such approximation.

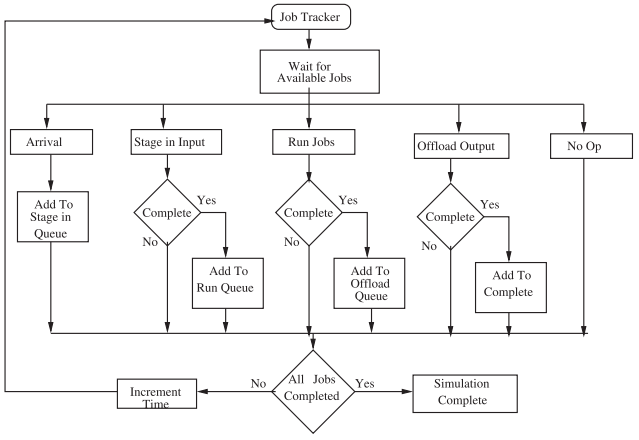

Fig. 7. Control flow in *simOffload*.

3) Simulator Output: simOffload provides an output trace with information about overall scratch space usage and the time it would take to offload the required data for a given job. This information can then further be used to determine any delay in meeting job scheduling deadlines.

4.1 Flow of Control in *simOffload*

simOffload maintains a pool of nodes arranged in a configurable topology to use as intermediate nodes. Nodes are randomly selected to facilitate the simulated offload. If a node is used for multiple offloads at the same time, the bandwidth is equally divided between the offloads. Moreover, *simOffload* can also capture varying storage capacities of the nodes and can alter offloading paths based on the capacities. In *simOffload*, we are mainly concerned with moving the data from the center to the first-level intermediate nodes only. Note that, while this can be easily extended to capture the end-user data delivery, we do not, as we can utilize our real implementation to more accurately study such behavior.

Fig. 7 illustrates *simOffload*'s operation. The main driver is a *Job tracker* that reads the logs, and selects an appropriate action for the simulator to take. We have opted for using the same time scale as the logs. At each job arrival, the tracker places it in a *wait queue*. The job input data staging is then started. The staging process may take many simulator ticks depending on the size of the input data, but once the process completes the job is moved to a *run queue*. The job will wait there until sufficient compute resources to run the job become available. Once the job completes its execution, it moves to the offload queue. If the simulator is modeling a decentralized offload, intermediate nodes will be chosen and the offload process will begin. If the standard approach is used, the data will remain on the scratch until it is purged by the center. Finally, *simOffload* also provides accounting and statistics about the offload process, such as the scratch space used and the data read, as well as other vital statistics.

TABLE 3
Interlevel Bandwidth Statistics

From	To	Average BW (Mbps)	Observed Stdev (Mbps)
Center	Level-1	20.7	16.7
Center	N_s	2.05	-
Level-1	Level2	5.4	3.6
Level-1	N_s	2.2	0.2
Level-2	N_s	8.4	12.9

5 Evaluation

We evaluate our result data offloading service using both the implementation of Section 3, and the HPC center data-subsystem simulator of Section 4. The goal of the evaluation is to show the effectiveness of our approach to better handle data offloading.

5.1 Implementation Results

We emulated the dynamic behavior of the proposed data offload model using the distributed test bed facilities of PlanetLab [13]. For our experiments, we chose 22 PlanetLab sites such that the HPC center and the submission site were on opposites coasts of the US, while the rest of the nodes were geographically scattered in between. All the nodes were arranged in a tree with the HPC center as root, the number of children ranging from zero to four, and two levels of intermediate nodes. Such a tree offers multiple data flow paths from the center to the submission site and allows for testing the approach under different scenarios. Table 3 shows the observed average bandwidths between the center, Level-1, Level-2, and N_s nodes used in our experiments. A more detailed description of our experimental setup can be found in [37]. In the following experiments, the chunk size was set to 256 kB. Moreover, the reported numbers represent averages over a set of three runs.

5.1.1 Approach Feasibility

In the first set of experiments, we determined the feasibility of our approach compared to several point-to-point direct transfer tools that are prevalent in HPC:

1. scp, a baseline secure transfer protocol;
2. IBP [38], an advanced transfer protocol that makes storage part of the network and allows programs to allocate and store data in the network near, where they are needed;
3. GridFTP [8], an extension to the FTP protocol, which provides authentication, parallel transfers and allows TCP buffer size tuning for high performance;
4. BBCP [39], which also provides high performance through parallel transfers and TCP buffer tuning. Note that these protocols are all typically supported [40] by HPC centers such as Jaguar [14].

We used a range of file sizes from 100 MB to 5.0 GB and measured the time for each direct transfer method between the center and the submission site. For our offloading, we used a combination of BitTorrent and NWS as outlined earlier.

In Table 4, we compare direct transfers with the times to offload data from the source (HPC center) to Level-1 nodes (Offload), time to forward the data from Level-1 to Level-2 (Push), and the time it takes the submission site to pull the data (Pull). Compared to the direct transfer mechanisms, the Offload is able to release the HPC center scratch space dramatically sooner for the data sizes we considered, as shown in Table 4. This has a significant impact on the HPC center serviceability since the free space can now be used for new incoming jobs.

Compared to each direct transfer mechanism, the time to pull the data to the submission site is also reduced as seen in Table 4. The reported pull time represents the time to transfer the file from Level-1 and Level-2 nodes to the submission site, and does not include the transfer time from the source. However, the submission site pull is asynchronous, and can start as soon as chunks begin to arrive at Level-1 nodes. We note that the overall transfer time, i.e., the time from when the source starts sending the data to when the submission site has received all the data is not a suitable metric, as our approach allows the site to be offline during the offloading process and delay starting the pull as necessary. However, the earliest time the user can get the output data is still a useful metric. In our system, the end user can start retrieving the data as soon as the center has offloaded it to Level-1 nodes. Thus, the Offload times reported in Table 4 also serve as the earliest data availability metric, and as stated earlier are significantly better in our approach compared to a direct data transfer.

5.1.2 Dynamic Data Scheduling

In this section, we compare our approach with a regular BitTorrent-based data transfer. In this case, we use NWS bandwidth measurements to greedily provision Level-1 nodes to increase the fan out until a maximum (predetermined) center outbound bandwidth is utilized.

TABLE 4
Comparison of Decentralized Transfer Times (in Seconds) with Different Direct Transfer Techniques

File Size		100 MB			240 MB			500 MB			2.1 GB			5.0 GB		
Decent-ralized	Offload	38			95			169			570			1339		
	Push	82			179			349			1123			2692		
	Pull	29			93			202			562			1387		
			Slowdown wrt.			Slowdown wrt.			Slowdown wrt.			Slowdown wrt.			Slowdown wrt.	
			Offld.	Pull		Offld.	Pull		Offld.	Pull		Offld.	Pull		Offld.	Pull
Direct	scp	286	7.5x	9.9x	727	7.7x	7.8x	1443	8.5x	7.1x	5834	10.2x	10.4x	13917	10.4x	10.0x
	IBP	183	4.8x	6.3x	431	4.5x	4.6x	929	5.5x	4.6x	3660	6.4x	6.5x	8546	6.4x	6.2x
	GridFTP	78	2.1x	2.7x	160	1.7x	1.7x	359	2.1x	1.8x	1603	2.8x	2.9x	3624	2.7x	2.6x
	BBCP	63	1.6x	2.2x	142	1.5x	1.5x	273	1.6x	1.4x	995	1.7x	1.8x	2373	1.8x	1.7x

The buffer size for IBP, GridFTP, and BBCP is set to 1 MB. The number of streams in GridFTP and BBCP is set to 8 and 16, respectively.

TABLE 5
The Time to Transfer a 5.0 GB File Using Standard BitTorrent

Phase	Time(s)
Send one copy from center (Offload)	2844
Send to all intermediate nodes (Push)	3684
Submission site download (Pull)	1393

The equivalent phases for our scheme are shown in brackets.

TABLE 6
Relative Improvement in File Transfer Times Using BitTorrent under Varying Chunk Sizes, Compared to the Default Chunk Size of 256 kB

Chunk Size	128 KB	256 KB	512 KB	1024 KB
Time saved (%)	-2.14	0	5.46	6.58

Table 4, discussed in the previous section, shows data offloading using the bandwidth measurement-based approach. Table 5 shows the time taken to deliver a 5.0 GB data set using the regular, unmodified BitTorrent protocol. Our results indicate that all three steps in our approach: Offload, Push, and Pull outperform the corresponding steps in regular BitTorrent transfer. The Offload from the HPC center to Level-1 nodes is 52.9 percent faster, while the Push from Level-1 nodes to Level-2 nodes is 26.9 percent faster. Use of bandwidth measurements, therefore, results in reduced intermediate forwarding time. The time to pull the file to the submission site is slightly increased by 0.4 percent. This is expected, as the flow paths do not affect the time it would take for the submission site to pull the file. These results show that bandwidth measurement provides a good tool for improving offload times.

5.1.3 Effect of Chunk Size on Offload Times

In our next experiment, we varied the chunk size used by BitTorrent, and observed the effects on file transfer time. The results are shown in Table 6. As the chunk size increases, the transfer time decreases. A chunk size of 1,024 kB improves transfer speed by 6.58 percent, when compared with the default chunk size of 256 kB. These results indicate that the transfers can benefit from larger chunk sizes.

5.1.4 When to Employ Staged Offload?

In the experimental setup that we have adopted, the bandwidth available between the center and Level-1 nodes is greater than that between the center and N_s. Thus, in this setup, the center will always decide to perform staged offloading. In the next experiment, we modified the setup to use a node from our Level-1 nodes, i.e., a node with better connectivity to the center, as the end user site, and did not use N_s. Then, we repeated the above experiment to offload a 2.1 GB file, first, without considering direct transfer and always using the staged offload mechanisms, and second, with the ability to choose between direct and staged offload depending on the ability to meet a SLA deadline. We observed that for the first case, the time to offload and pull the data was 610 s and 400 s, respectively. In contrast, for the second case the direct transfer completed in 380 s, an improvement of 37.7 percent in offload times. This result coupled with the earlier experiments stress the need for the offload mechanisms to dynamically adjust to the variations in the system behavior and to not be hard wired to simply always do a staged offload or a direct transfer.

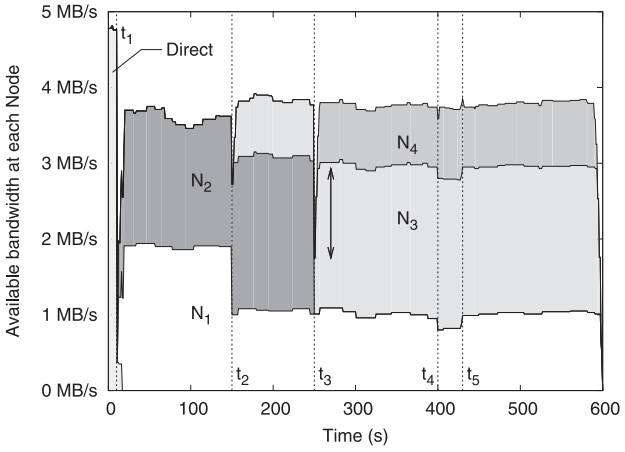

Fig. 8. Utilized out-bound bandwidth at the center, as the system adjusts to failures and meets the 600 s deadline for offloading. The labeled regions represent utilized bandwidth to individual nodes.

5.1.5 Enforcing SLA

In the next experiment, we study the effectiveness of the proposed approach in enforcing SLAs. We assume that the submission site and the HPC center have agreed on an SLA to offload the 2.1 GB file to four Level-1 nodes (N_1 to N_4) or a direct transfer in 600 s. Initially, we choose a site that supports a large bandwidth between the center and the site. Thus, our algorithm starts off by doing a direct transfer. However, at time $t_1 = 10$ s, we limit the inbound bandwidth of the site to 1/10 of its value. Soon after this happens, our system realizes that the SLA cannot be met with a direct transfer and switches to a staged offload. Once an offload schedule is chosen, we utilize bandwidth provided by the NWS to estimate the time E_t it would take to offload the remaining chunks of the file. If E_t turns out to be longer than necessary to meet the SLA, the fan out is increased. The process is repeated every time the available bandwidth predictions change. To force dynamic scheduling to come into play, we artificially introduced two bandwidth-changing events during the offload: at time $t_2 = 150$ s, we limited the available bandwidth to N_1 to about 1 MB/s; and at time $t_3 = 250$ s, we failed N_2. Fig. 8 shows the sum of the utilized bandwidths between the center and each of the four Level-1 nodes reported every second. Initially, only N_1 and N_2 are used. Soon after t_2, the drop in N_1's bandwidth is detected causing an increase in E_t. The system reacts by increasing the fan out to use N_3, so that E_t remains under the 600 s deadline. Note that between t_2 and t_3, the maximum available bandwidth of N_3 was not needed to meet the SLA and was not utilized. However, when N_2 failed at t_3, the system first uses N_3's maximum bandwidth as observed as a spike (indicated by the arrow) in N_3's curve following t_3. However, this increase is not sufficient to compensate for the loss of N_2, hence, the fan out is adjusted to also use N_4. Also note that between t_4 and t_5, the available bandwidth for N_1 is reduced significantly enough to cause the system to utilize a higher bandwidth to N_4, so that the overall total bandwidth is maintained to meet the SLA. Once N_1's bandwidth returns too normal, our greedy algorithm once again increases the use of N_1's bandwidth

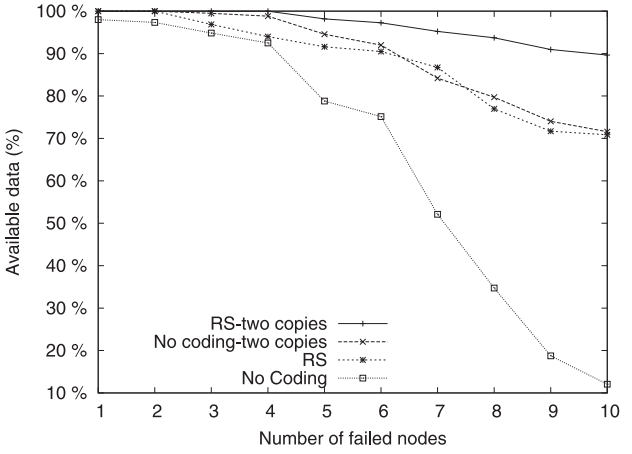

Fig. 9. Available data under different error coding schemes, as intermediate nodes fail.

and reduces the use of N_4's bandwidth. The two spikes at t_4 and t_5 capture the system response time to these events. Finally, as observed from the figure, the system is able to transfer the file within the specified SLA by dynamically adjusting the fan out.

5.1.6 Data Availability

In this experiment, we measured the effect of error coding in achieving fault tolerance. For this purpose, we randomly failed several intermediate nodes during the course of the transfer and determined what portions of the file have become unavailable. The experiment was repeated with increasing number of failed nodes, up to 10 (50 percent). Fig. 9 shows the average results over three runs for four scenarios: with no error coding, using 4:5 RS [35] coding, and using replication to create two copies under both no error coding and RS. As expected, using neither error coding nor replication causes data to become unavailable even with a single failure, with up to 87.9 percent data being unavailable with 10 failed nodes. Use of error coding or replication allows the file to be transferred successfully even when multiple nodes on the path from the center to the client fail. Note that both RS-single copy and replication are able to provide 100 percent availability with up to two (10 percent) node failures. This is promising as our RS code have only 25 percent redundancy to that of 100 percent of replication. However, with additional node failures simple replication is able to provide better availability than RS. Creating two copies of data under RS further improves data availability: 100 percent availability when 25 percent of the intermediate nodes have failed, 89.7 percent availability with the extreme case of 50 percent of failed intermediate nodes. Hence, error coding at the center along with replication through multiple data-flow paths can provide excellent fault-tolerance behavior for the offloading process.

5.2 Simulation Results

In the next set of experiments, we utilized our simulator (see Section 4) to study in detail the impact of our approach on overall scratch utilization, and toward mitigating the role of failures in job scheduling delays.

TABLE 7
Statistics about the Job Logs Used in This Study

Duration	22764 Hrs
Number of jobs	80234
Job execution time	30 s to 120892 s, average 5835 s
Data size	2.28 MB to 3714 GB, average 32.1 GB

5.2.1 Center Log Statistics

The simulator is driven by job-statistics logs collected over a period of three-years (2004-2007) on the Jaguar [14] supercomputer. Table 7 shows some relevant characteristics of the logs. Also, note the large variance in both the duration of the jobs (from a few seconds to over a day) and the amount of data they access (from a few MBs to several TBs), implying that even a small amount of scratch savings for larger jobs can enable accommodating a large number of smaller jobs, consequently increasing the center's job throughput. For this study, we assume that the job input and output data sizes are capped to the total aggregate memory usage of the job. For example, if the job used 1,000 compute cores and 2 GB of memory per core, we assume its output data size to be 2,000 GB. This is a very reasonable assumption given that many data-intensive applications' checkpoint or restart output data sets cannot be larger than their total memory usage. In the absence of per job output data size information in the logs, we consider such an estimate to be a realistic approximation, capturing current usage trends. The output data itself can run in the hundreds of GBs and TBs for leadership simulations on Jaguar. For example, Fusion applications such as GTC, GTS, and XGC1 produce 44 TB, 50 TB, and 300 GB, respectively, of output data from runs on 100,000+ cores. Given that outputs themselves can be quite large, we did not include the effect of intermediate checkpoint snapshot data on scratch utilization.

5.2.2 Impact on Scratch Space Utilization

In the first set of experiments, we quantify the impact of our timely offloading approach on scratch space usage. We play the logs in our simulator and determine the amount of scratch used both under a 7-day purge policy and decentralized offloading. For this test, we assume that the scratch is empty at the beginning. Only output data is considered, and a data item is only purged if its associated job has completed. Fig. 10 shows the scratch space usage under the studied approaches, measured every 10 minutes. Observe that the scratch utilization under decentralized offloading is (as much as an order of magnitude) lower than that under a 7-day purge.

To further illustrate the reduced scratch usage, Fig. 11 shows the average per hour savings achieved by the decentralized offloading approach. Here, we observe that, on average, across the entire log, decentralized offloading uses 88.2 percent less scratch per unit of time (e.g., 882 GB/Hr, on average, per Terabyte of storage) compared to a simple 7-day purge. Thus, our approach is a promising way for conserving precious scratch resource.

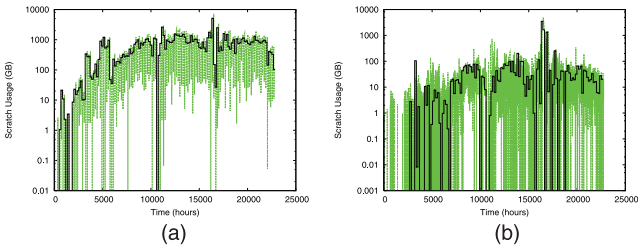

Fig. 10. Overall scratch utilization over the duration of the traces under different approaches. The solid line shows the average utilization measured per hour. (a) 7-day purge. (b) Decentralized offload.

5.2.3 Impact on Job Scheduling and Center Serviceability

In the next experiment, we limit the available scratch space, and study how job scheduling will be affected under a simple 7-day purge and our decentralized offloading with several redundancy improving techniques, namely, erasure coding, two data copies, and two-copies plus erasure coding. To analyze the impact of completed jobs' data offloading on the scheduling of new incoming jobs, we measure the delay that might be incurred in starting the new jobs. New jobs will be delayed if their input data cannot be staged into the scratch space due to a lack of sufficient space, resulting from the scratch not having been cleared of result output data from the previously completed jobs. Table 8 shows the results in terms of the number of jobs delayed, the maximum observed delay, as well as the average delay. Compared to a 7-day purge, decentralized offloading can significantly reduce the delays: 78.1 percent, and 98.6 percent for 2.5 TB and 1 TB scratch size, respectively, while 5 TB under decentralized offloading experiences no delays. Moreover, introducing redundancy improving techniques also introduce delays, however, such delays are nominal compared to the 7-day purge. For instance, the average delay under decentralized offloading with both erasure coding and two-copies is 85.4 percent, 45.7 percent, and 97.7 percent less than that under 7-day purge for a scratch size of 5 TB, 2.5 TB, and 1 TB, respectively.

Next, we calculate the impact of observed delays in job scheduling using a new metric, *Expanded Usage Factor* (EUF), which we define as the ratio *(execution_time + data_wait_time)/execution_time*, where the *data_wait_time* is the time the output data has to wait on the scratch space after job completion before being offloaded. Our EUF metric is inspired by the widely used *expansion factor* [3], [14], which is often used to quantify job delays in HPC centers. Expansion factor is defined as the ratio *(wall_time + wait_time)/wall_time* averaged over all jobs (the closer to 1, the better). Similarly, EUF indicates the extra, avoidable time for which a data set occupies the scratch space, and the closer its value is to one, the better. Thus, EUF also provides a valuable measure for the HPC center serviceability in terms of precious scratch space consumption. Fig. 12 shows the average EUF for the HPC center, for a duration of three years and 80,234 jobs, under different scratch sizes. Once again, the decentralized offloading (even with erasure coding and two data copies) behaves superior to a 7-day purge with average EUF reduction of 99 percent, 94 percent, and 97 percent, with available scratch size of 1 TB, 2.5 TB, and 5 TB, respectively. Observe that even when the scratch space is 5 TB, i.e., well beyond the job-trace footprints, the decentralized approach provides a much better EUF.

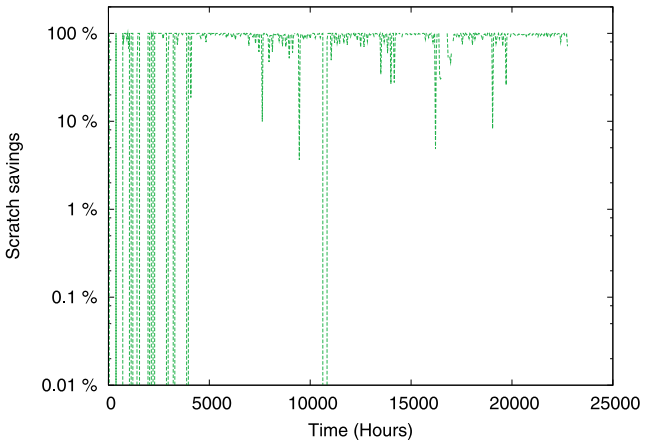

Fig. 11. The scratch savings achieved by using a decentralized offload compared to a standard 7 day purge.

5.2.4 Impact of Failures

Next, we measure how first-level intermediate node failures during the decentralized offloading process impact job scheduling. The total number of intermediate nodes used in this study is 25, and we assume that 10 percent to 50 percent of these nodes fail randomly during the course of an offload. Table 9 shows the corresponding delays under the studied scenarios. It is observed that compared to the case of no failures, intermediate-node failures can significantly delay job scheduling. However, as the failures increase from 10 percent to 50 percent, the average delay remains under 68.3 percent, 24.5 percent, and 25.3 percent for 5 TB, 2.5 TB, and 1 TB scratch size, respectively.

Next, Fig. 13 shows how first-level intermediate node failures affect the EUF. A large number of failures of the first-level nodes may result in retransmission to ensure data is not lost, which in turn may cause the offload process to

TABLE 8
Job Delays under the Different Offloading Approaches

Offload Type	Number of Jobs Delayed			Maximum Delay (Hrs)			Average Delay (Hrs)		
	5.0 TB	2.5 TB	1.0 TB	5.0 TB	2.5 TB	1.0 TB	5.0 TB	2.5 TB	1.0 TB
7-day Purge	2541	11027	71253	59.0	351.3	7347.2	37.2	109.3	3446.0
Decent. Offloading (DO)	0	1010	2893	0.0	38.6	92.0	0.0	24.0	47.4
DO + Encoding	0	1226	3013	0.0	58.3	113.7	0.0	42.6	57.7
DO + 2 copies	114	1874	3409	2.5	82.0	142.2	1.8	54.2	77.0
DO + Encoding + 2 copies	197	1739	3207	7.6	85.8	142.0	5.4	59.2	80.8

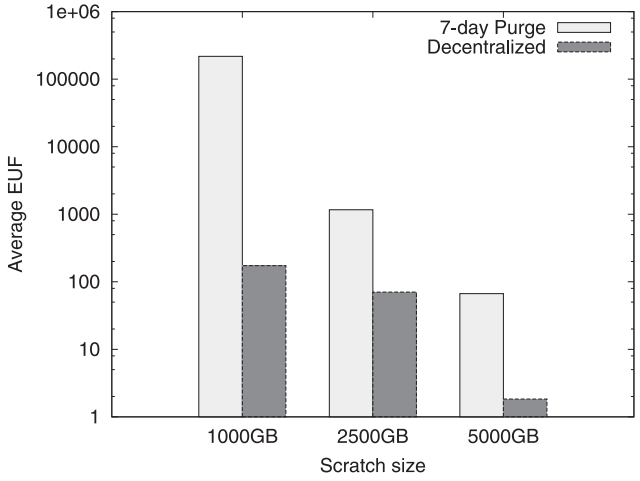

Fig. 12. Average EUF for the HPC center for a duration of three years and 80,234 jobs, under a 7-day purge and decentralized offloading with erasure coding and two-copies for varying scratch sizes.

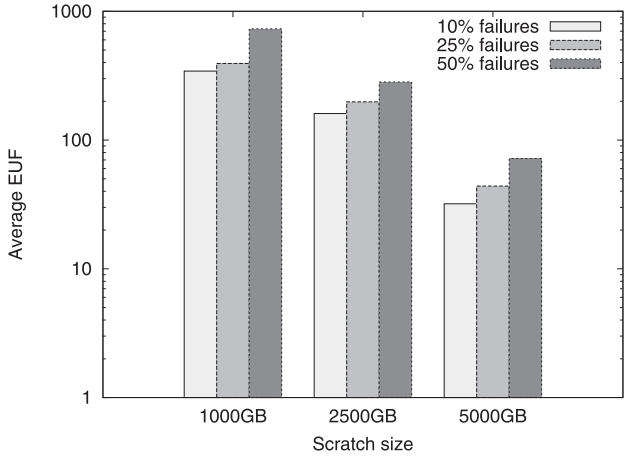

Fig. 13. Average EUF under decentralized offloading with replication and erasure coding for varying scratch sizes and different intermediate node failure rates.

take longer. It is observed that with a constrained scratch size of 1 TB, average EUF is increased by 98 percent, 127 percent, and 323 percent for 10 percent, 25 percent, and 50 percent of the nodes failing compared to the no failure case of Fig. 12. Similar failure affects are observed for other scratch sizes.

In summary, the decentralized offloading approach is promising in its ability to reduce job scheduling delays, improving expansion factor, and can tolerate failures without drastically degrading overall system performance.

6 Related Work

HPC data management is a critical research area, and a number of works have explored it from different perspectives. In the following, we discuss several related works.

The use of intermediate buffers to hide latency or to provide fault tolerance is a common practice in OS as well as file systems. Kangaroo [41] extends this idea to Grid computing, with the goal to provide reliability against transient resource availability. It hides network storage using an application perceived file system with relaxed consistency semantics. However, Kangaroo simply provides a staged transfer mechanism and does not concern itself with network vagaries or changing route dynamics in an end-to-end data path.

IBP [38] offers a data distribution infrastructure with a set of strategically placed resources, storage depots, to move data. Together with the transport protocol, this is referred to as logistical networking. We differ in our approach to combine both a staged as well as a decentralized data delivery. The induction of user-specified nodes also allows the system to optimize the offload on a per-user basis, which is not possible with IBP. Further, our approach is unique as we strive to meet a deadline in data delivery and offload from the HPC center.

In [42], the authors stream outputs from GTC runs through logistical networking. The adaptive buffer strategy reconciles the rate of data production with that of available network resources by failing over the transfer to a local IBP depot in the case of a network failure. The goal here is to overlap computation with in-situ network data transfer. This is complementary to our work and such in-situ processing can also benefit from our decentralized transport.

Timely offloading of HPC center data can only be achieved by coinciding the output data movement with the completion of the compute job. Our previous work in this regard treats data offload as an I/O job and schedules it alongside computation so it begins at job completion [17]. Techniques presented in this paper complement the coordinated scheduling and the two approaches are used together as detailed in the architecture.

Stork [43] a scheduler for data placement activities in a grid environment, along with Condor [44] and DAGMan [45] is used to schedule data and computation together in the face of vagaries. However, these systems are positioned as a part of the application workflow rather than a set of HPC center integrated services, where our work resides.

DMOVER [46] is a tool, that is, used for moving data in the TeraGrid by aggregating data transfer commands in a script and scheduling them using a separate queue. However, it only addresses point-to-point data transfers using GridFTP. In contrast, our work achieves a decentralized data delivery and coincides it with job completion.

TABLE 9
Observed Job Delays under Decentralized Offloading with Erasure Coding and Two-Copies, when 10, 25, or 50 Percent of the First-Level Intermediate Nodes have Failed

Percent Failed	Number of Jobs Delayed			Maximum Delay (Hrs)			Average Delay (Hrs)		
	5.0 TB	2.5 TB	1.0 TB	5.0 TB	2.5 TB	1.0 TB	5.0 TB	2.5 TB	1.0 TB
10 percent	1129	2613	4991	66.9	156.8	221.0	42.9	90.2	96.0
25 percent	1217	2687	5405	82.1	185.1	248.4	54.0	108.3	101.4
50 percent	1484	3553	7059	122.9	231.4	325.5	72.2	112.3	120.9

Our solution can also be built upon to transfer intermediate checkpoint data as long as our system is notified about the availability of the data.

A number of systems such as Bullet [47], [48], Shark [49], CoDeeN [50], and CoBlitz [51] have explored the use of multicast and p2p-techniques for transferring large amounts of data between multiple Internet nodes. Our work requires factoring in center-user service agreements and dynamic resource availability, which are not considered in the above systems.

Content distribution networks (CDN) such as CoDeeN [50] effectively implement a system of proxy servers that users can explicitly use for faster delivery of data to their nodes. FastReplica [52] creates replicas of data on different CDN nodes to support faster data access. Our work shares the multicast techniques but differs in automatic dynamic selection of intermediary nodes to facilitate multicast when necessary.

A number of bulk data-transfer protocols have been developed for Internet use, e.g., Slurpie [53] allows clients to simultaneously contact a server and use random back-off to avoid performance degradation due to congestion. The approach of downloading large files from several mirror sites has been validated by its wide-spread use in BitTorrent [16], and many protocols for parallel downloading from mirror sites have been proposed [54], [55], [56]. These works are complimentary, and we built on the principles developed in these systems, especially BitTorrent.

The NWS [12] provides a powerful framework, which allows the resources of distributed computers to be monitored. NWS bandwidth measurements have been used in a static context to determine a Grid data site, offering optimal download rates, from among multiple replicated alternatives [57]. In this work, however, we use measurements to determine a path within a network of nodes and dynamically adjust it based on bandwidth degradation.

The GridFTP overlay network service [58] implements a specialized data storage interface (DSI) to achieve split-TCP functionality. The GridFTP client command is issued with source and destination URLs A/C and C/D to denote a transfer between end points A and D through nodes C and D. In [59], the authors have extended this effort to use previous transfers as a measure to use a particular node in the transfer overlay. Our work differs as it delivers data on a user-specified deadline and further uses dynamic measurements to adapt and adjust the fan out of transfers.

Finally, staging of data on the HPC center to enable timely execution of jobs is another related direction. Recent work on staging by others [60] as well as our own [36], [61] has shown the importance of integrating staging services into center management software. Such works are complementary to our work, and vice versa, in that an integrated staging and offloading solution can significantly improve the overall serviceability of a center.

7 Conclusion

In this paper, we have presented the design and implementation of a result-data offloading service for HPC centers. Offloading large data to end-user locations in a timely manner is critical to center operations, its availability and serviceability. Our approach presents a fresh look at offloading by using a set of user-specified intermediate nodes to construct a p2p network and transferring data based on bandwidth adaptation.

Our results indicate that our offloading approach improves the rate at which the data is offloaded from the center (90.4 percent for a 5 GB data transfer), while allowing the submission site to pull the data as and when the site becomes available, at a much higher transfer rate because the result-data has already been staged closer. Further, offloading enables us to deliver data based on a previously agreed upon SLA, dynamically varying the fan-out as necessary. An analysis of our approach using a realistic simulator, *simOffload*, driven by a three-year log from an actual supercomputer reveals that it is better able to manage the scratch space and reduce job delays. Thus, our scheme can be extremely useful to both HPC centers and users.

Our evaluation shows that the presented offloading scheme reacts well to system variations in meeting user-center SLA's and deciding when a staged offload is preferable to a direct transfer, and achieves good fault tolerance via its use of erasure coding and replication.

In summary, the offloading approach effectively utilizes orthogonal, residual bandwidth and can serve as an alternative to direct transfers, which may not always be feasible, optimal, or fault-tolerant. Moreover, this approach allows for a more integrated HPC center management solution than the extant ad hoc techniques. Finally, while distributed offloading is highly competitive, it raises new research questions in terms of the strategic placement, and selection, of intermediate nodes between an HPC center and end-user destinations.

Acknowledgments

This research is sponsored in part by the LDRD program of ORNL, managed by UT-Battelle, LLC for the US Department of Energy under Contract No. DE-AC05-00OR22725, and by the US National Science Foundation grant CCF-0746832.

References

[1] Gyrokinetic Toroidal Code (GTC), http://gk.ps.uci.edu/GTC/index.html, 2010.
[2] W.X. Wang, Z. Lin, W.M. Tang, W.W. Lee, S. Ethier, J.L.V. Lewandowski, G. Rewoldt, T.S. Hahm, and J. Manickam, "Global Gyrokinetic Particle Simulation of Turbulence and Transport in Realistic Tokamak Geometry," *J. Physics: Conf. Series*, vol. 16, no. 1, p. 59, 2005.
[3] NSF TeraGrid, http://www.teragrid.org/, 2009.
[4] Cluster File Systems, Inc., Lustre: A Scalable, High-Performance File System, http://www.lustre.org/-docs/-whitepaper.pdf, 2002.
[5] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proc. USENIX Conf. File and Storage Technologies (FAST '02)*, 2002.
[6] NCCS.GOV File Systems, http://info.nccs.gov/computing-resources/jaguar/file-systems, 2007.
[7] UC/ANL Teragrid Guide, http://www.uc.teragrid.org/tg-docs/user-guide.html#disk, 2004.
[8] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Proc. Workshop I/O in Parallel and Distributed Systems (IOPADS '99)*, 1999.
[9] M. Gleicher, "HSI: Hierarchical Storage Interface for HPSS," http://www.hpss-collaboration.org/hpss/HSI/, 2010.

[10] J.W. Cobb, A. Geist, J.A. Kohl, S.D. Miller, P.F. Peterson, G.G. Pike, M.A. Reuter, T. Swain, S.S. Vazhkudai, and N.N. Vijayakumar, "The Neutron Science Teragrid Gateway: A Teragrid Science Gateway to Support the Spallation Neutron Source: Research Articles," *Concurrency and Computation: Practice and Experience,* vol. 19, no. 6, pp. 809-826, 2007.

[11] M. Christie and S. Marru, "The Lead Portal: A Teragrid Gateway and Application Service Architecture: Research Articles," *Concurrency and Computation : Practice and Experience,* vol. 19, no. 6, pp. 767-781, 2007.

[12] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computer Systems ,* vol. 15, no. 5, pp. 757-768, 1999.

[13] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. ACM First Workshop Hot Topics in Networks (HotNets-I),* 2002.

[14] Nat'l Center for Computational Sciences, http://www.nccs.gov/, 2009.

[15] A. Bayucan, R.L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, "Portable Batch System: External Reference Specification," 2672 Bayshore Parkway, Suite 810, Mountain View, CA 94043, http://www-unix.mcs.anl.gov/openpbs/docs/v2_2_ers.pdf, Nov. 1999.

[16] B. Cohen BitTorrent Protocol Specification, http://www.bittorrent.org/protocol.html, 2007.

[17] Z. Zhang, C. Wang, S.S. Vazhkudai, X. Ma, G. Pike, J. Cobb, and F. Mueller, "Optimizing Center Performance through Coordinated Data Staging, Scheduling and Recovery," *Proc. Conf. Supercomputing,* 2007.

[18] Dept. of Energy, Office of Science, Innovative and Novel Computational Impact on Theory and Experiment (INCITE), http://www.er.doe.gov/ascr/incite/, 2008.

[19] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing Force Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Middleware,* 2001.

[20] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM,* 2001.

[21] Internet2, http://www.internet2.edu/, 2008.

[22] Nat'l Lambda Rail: Light the Future, http://www.nlr.net/, 2008.

[23] The Research and Education Data Depot Network (Reddnet), http://www.lstore.org/pwiki/pmwiki.php?n=REDDnet. Infrastructure, 2007.

[24] Grid Physics Network, http://www.griphyn.org, 2004.

[25] P. Maymounkov Online Codes, Technical Report TR2003-883, New York Univ., Nov. 2002.

[26] J.S. Plank, "Erasure Codes for Storage Applications," Tutorial Slides, Presented at USENIX FAST, http://www.cs.utk.edu/plank/plank/papers/FAST-2005.html, 2005.

[27] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an Mttf of 1,000,000 Hours Mean to You?," *Proc. USENIX Conf. File and Storage Technologies (FAST '07),* 2007.

[28] E. Pinheiro, W.-D. Weber, and L. André Barroso, "Failure Trends in a Large Disk Drive Population," *Proc. USENIX Conf. File and Storage Technologies (FAST '07),* 2007.

[29] S. Shah and J.G. Elerath, "Reliability Analysis of Disk Drive Failure Mechanisms," *Proc. IEEE Ann. Reliability and Maintainability Symp. (RAMS '05),* 2005.

[30] L.N. Bairavasundaram, G.R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," *Proc. ACM SIGMETRICS,* 2007.

[31] I. Iliadis, R. Haas, X.-Y. Hu, and E. Eleftheriou, "Disk Scrubbing versus Intra-Disk Redundancy for High-Reliability Raid Storage Systems," *Proc. ACM SIGMETRICS,* 2008.

[32] A. Riska and E. Riedel, "Idle Read after Write: Iraw," *Proc. USENIX Ann. Technical Conf. (ATC '08),* 2008.

[33] A.R. Butt, T.A. Johnson, Y. Zheng, and Y. Charlie Hu, "Kosha: A Peer-to-Peer Enhancement for the Network File System," *J. Grid Computing: Special Issue on Global and Peer-to-Peer Computing,* vol. 4, no. 3, pp. 323-341, 2006.

[34] Druschel et. al. Freepastry, http://freepastry.rice.edu/, 2004.

[35] J.S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," *Software—Practice and Experience,* vol. 27, no. 9, pp. 995-1012, 1997.

[36] H. Monti, A.R. Butt, and S.S. Vazhkudai, "/Scratch as a Cache: Rethinking HPC Center Scratch Storage," *Proc. ACM Ann. Int'l Conf. Supercomputing (ICS '09),* 2009.

[37] H. Monti, A.R. Butt, and S.S. Vazhkudai, "Timely Offloading of Result-Data in Hpc Centers," *Proc. ACM Ann. Int'l Conf. Supercomputing (ICS '08),* 2008.

[38] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski, "The Internet Backplane Protocol: Storage in the Network," *Proc. Network Storage Symp. (NSS '99),* 1999.

[39] Bbcp Homepage, http://www.slac.stanford.edu/, 2010.

[40] Nccs User Support—Data Transfer, http://www.nccs.gov/user-support/general-support/data-transfer/, 2010.

[41] D. Thain, S. Son, J. Basney, and M. Livny, "The Kangaroo Approach to Data Movement on the Grid," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC '01),* 2001.

[42] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. Mccune, and M. Parashar, "High Performance Threaded Data Streaming for Large Scale Simulations," *Proc. IEEE/ACM Int'l Workshop Grid Computing,* 2004.

[43] T. Kosar and M. Livny, "Stork: Making Data Placement a First Class Citizen in the Grid," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '04),* 2004.

[44] M. Litzkow, M. Livny, and M. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '88),* 1988.

[45] Directed Acyclic Graph Manager, http://www.cs.wisc.edu/condor/dagman/, 2010.

[46] DMOVER: Scheduled Data Transfer for Distributed Computational Workflows, http://www.psc.edu/general/software/packages/dmover/, 2008.

[47] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *Proc. ACM Symp. Operating Systems Principles (SOSP '03),* 2003.

[48] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A.M. Vahdat, "Using Random Subsets to Build Scalable Network Services," *Proc. Conf. USENIX Symp. Internet Technologies and Systems (USITS '03),* 2003.

[49] S. Annapureddy, M.J. Freedman, and D. Mazires, "Shark: Scaling File Servers via Cooperative Caching," *Proc. Conf. USENIX Networked Systems Design and Implementation (NSDI '05),* 2005.

[50] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and Security in the CoDeeN Content Distribution Network," *Proc. USENIX Ann. Technical Conf. (ATC '04),* 2004.

[51] K. Park and V.S. Pai, "Scale and Performance in the CoBlitz Large-File Distribution Service," *Proc. Conf. USENIX Networked Systems Design and Implementation (NSDI '06),* 2006.

[52] L. Cherkasova and J. Lee, "Fastreplica: Efficient Large File Distribution within Content Delivery Networks," *Proc. Conf. USENIX Symp. Internet Technologies and Systems (USITS '03),* 2003.

[53] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," *Proc. IEEE INFOCOM,* 2004.

[54] P. Rodriguez, A. Kirpal, and E.W. Biersack, "Parallel-access for Mirror Sites in the Internet," *Proc. IEEE INFOCOM,* 2000.

[55] J.S. Plank, S. Atchley, Y. Ding, and M. Beck, "Algorithms for High Performance, Wide-Area Distributed File Downloads," *Parallel Processing Letters,* vol. 13, no. 2, pp. 207-224, 2003.

[56] R.L. Collins and J.S. Plank, "Downloading Replicated, Wide-Area Files—A Framework and Empirical Evaluation," *Proc. IEEE Int'l Symp. Network Computing,* 2004.

[57] S. Vazhkudai and J. Schopf, "Predicting Sporadic Grid Data Transfers," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC '02),* 2002.

[58] P. Rizk, C. Kiddle, and R. Simmonds, "A Gridftp Overlay Network Service," *Proc. Int'l Conf. Grid Computing,* 2007.

[59] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, I. Foster, and J. Saltz, "Using Overlays for Efficient Data Transfer over Shared Wide-Area Networks," *Proc. Int'l Conf. Supercomputing,* 2008.

[60] H. Abbasi, M. Wolf, F. Zheng, G. Eisenhauer, S. Klasky, and K. Schwan, "Scalable Data Staging Services for Petascale Applications," *Proc. ACM Int'l Symp. High Performance Distributed Computing (HPDC '09),* 2009.

[61] H. Monti, A.R. Butt, and S.S. Vazhkudai, "Just-in-Time Staging of Large Input Data for Supercomputing Jobs," *Proc. ACM Petascale Data Storage Workshop (PDSW '08),* 2008.

Henry M. Monti received the BS degree in computer science from George Mason University, in 2006. He is currently working toward the PhD degree in computer science and applications at Virginia Polytechnic Institute and State University. His research interests include HPC, distributed systems, and cloud computing. He is a student member of the IEEE and the IEEE Computer Society.

Ali R. Butt received the BSc (hons) degree in electrical engineering from the University of Engineering and Technology Lahore, Pakistan, in 2000 and the PhD degree in electrical and computer engineering from Purdue University, in 2006. He is an assistant professor of computer science at Virginia Tech. At Purdue, he also served as the president of the Electrical and Computer Engineering Graduate Student Association for 2003 and 2004. His research interests includes experimental computer systems, especially in data-intensive HPC and the impact of technologies such as massive multicores, cloud computing, and asymmetric architectures on HPC. His current work focuses on I/O and storage issues faced in modern HPC systems. He is a recipient of the NSF CAREER Award (2008), an IBM Faculty Award (2008), an IBM Shared University Research Award (2009), and a Virginia Tech College of Engineering "Outstanding New Assistant Professor" Award (2009). He was an invited participant (2009) and an organizer (2010) for the NAE's US Frontiers of Engineering Symposium. He is a member of the USENIX, ACM, and ASEE, and a senior member of the IEEE.

Sudharshan S. Vazhkudai received the doctorate degree, from the University of Mississippi, in 2003 and performed his research at Argonne National Laboratory. He is a research scientist in the computer science and mathematics division at Oak Ridge National Laboratory, US Department of Energy Facility. In addition, he is also a joint faculty associate professor at the University of Tennessee. He is broadly interested in storage systems, HPC I/O architectures, and distributed computing.